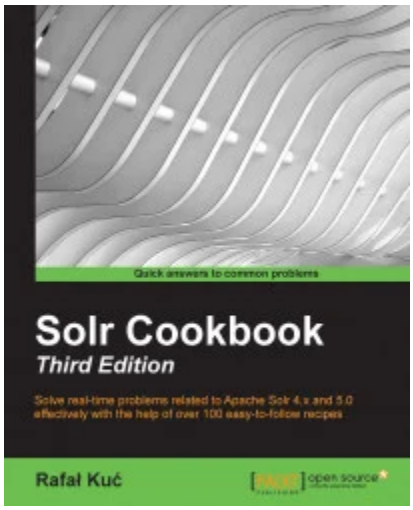
Solr Cookbook

subscription.packtpub.com/book/data/9781783553150/7



Chapter 7. In the Cloud

In this chapter, we will cover the following topics:

- Creating a new SolrCloud cluster
- Setting up multiple collections on a single cluster
- Splitting shards
- Having more than a single shard from a collection on a node
- Creating a collection on defined nodes
- Adding replicas after collection creation
- Removing replicas
- Moving shards between nodes
- Using aliasing
- Using routing

Introduction

With the release of Apache Solr 4.0, we were given a new, powerful mode Solr could work in —SolrCloud. What we got is out-of-the-box distributed indexing and searching at a full scale. We can distribute our collection along multiple machines without having to think about doing it in our application. We can have multiple logical collections defined, running, and managed automatically. In this chapter, we'll see how to manage your SolrCloud instances, how to increase the number of replicas, and have multiple collections inside the same cluster.

This chapter covers both Solr 4.x and Solr 5 when it comes to the creation of SolrCloud clusters and handling collections.

Creating a new SolrCloud cluster

Imagine a situation where one day you have to set up a distributed cluster with the use of Solr. The amount of data is just too much for a single server to handle. Of course, you can just set up a second server or go for another master server with another set of data. But before Solr 4.0, you would have to take care of the data distribution yourself. In addition to this, you would also have to take care of setting up replication, data duplication, and so on. With SolrCloud, you don't have to do this—you can just set up a new cluster, and this recipe will show you how to do that.

Getting ready

Before continuing further, I advise you to read the *Installing ZooKeeper for SolrCloud* recipe from Chapter 1, *Apache Solr Configuration*. It shows you how to set up a Zookeeper cluster in order to be ready for production use. However, if you already have Zookeeper running, you can skip this recipe.

How to do it...

Let's assume that we want to create a cluster that will have four Solr servers. We also would like to have our data divided between the four Solr servers in such a way that we have the original data on two machines, and in addition to this, we would also have a copy of each shard available in case something happens with one of the Solr instances. I also assume that we already have our Zookeeper cluster set up, ready, and available at the address `192.168.1.10` on the `9983` port. For this recipe, we will set up four SolrCloud nodes on the same physical machine:

1. We will start by running an empty Solr server (without any configuration) on port `8983`. We do this by running the following command (for Solr 4.x):

```
java -DzkHost=192.168.1.10:9983 -jar start.jar
```

2. For Solr 5, we will run the following command:

```
bin/solr -c -z 192.168.1.10:9983
```

3. Now we start another three nodes, each on a different port (note that different Solr instances can run on the same port, but they should be installed on different machines). We do this by running one command for each installed Solr server (for Solr 4.x):

```
java -Djetty.port=6983 -DzkHost=192.168.1.10:9983 -jar start.jar
java -Djetty.port=4983 -DzkHost=192.168.1.10:9983 -jar start.jar
java -Djetty.port=2983 -DzkHost=192.168.1.10:9983 -jar start.jar
```

4. For Solr 5, the commands will be as follows:

```
bin/solr -c -p 6983 -z 192.168.1.10:9983
bin/solr -c -p 4983 -z 192.168.1.10:9983
bin/solr -c -p 2983 -z 192.168.1.10:9983
```

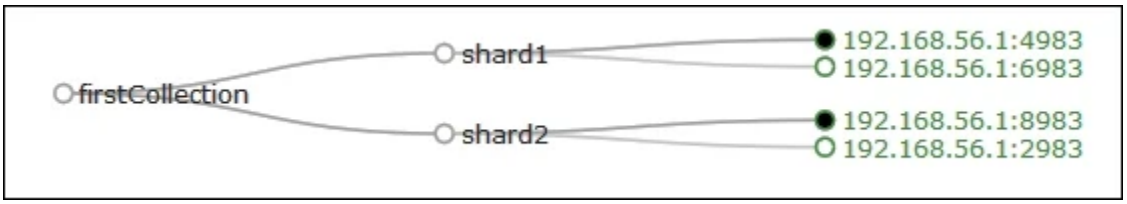
5. Now we need to upload our collection configuration to ZooKeeper. Assuming that we have our configuration in `/home/conf/solrconfiguration/conf`, we will run the following command from the `home` directory of the Solr server that runs first (the `zkcli.sh` script can be found in the Solr deployment example in the `scripts/cloud-scripts` directory):

```
./zkcli.sh -cmd upconfig -zkhost 192.168.1.10:9983 -confdir
/home/conf/solrconfiguration/conf/ -confname collection1
```

6. Now we can create our collection using the following command:

```
curl 'localhost:8983/solr/admin/collections?
action=CREATE&name=firstCollection&numShards=2&replicationFactor=2&collection.c
```

7. If we now go to `http://localhost:8983/solr/#/~cloud`, we will see the following cluster view:



As we can see, Solr has created a new collection with a proper deployment. Let's now see how it works.

How it works...

We assume that we already have ZooKeeper installed—it is empty and doesn't have information about any collection, because we didn't create them.

For Solr 4.x, we started by running Solr and telling it that we want it to run in SolrCloud mode. We did that by specifying the `-DzkHost` property and setting its value to the IP address of our ZooKeeper instance. Of course, in the production environment, you would point Solr to a cluster of ZooKeeper nodes—this is done using the same property, but the IP addresses are separated using the comma character.

For Solr 5, we used the `solr` script provided in the `bin` directory. By adding the `-c` switch, we told Solr that we want it to run in the SolrCloud mode. The `-z` switch works exactly the same as the `-DzkHost` property for Solr 4.x—it allows you to specify the ZooKeeper host that should be used.

Of course, the other three Solr nodes run exactly in the same manner. For Solr 4.x, we add the `-DzkHost` property that points Solr to our ZooKeeper. Because we are running all four nodes on the same physical machine, we needed to specify the `-Djetty.port` property,

because we can run only a single Solr server on a single port. For Solr 5, we use the `-z` property of the `bin/solr` script and we use the `-p` property to specify the port on which Solr should start.

The next step is to upload the collection configuration to ZooKeeper. We do this because Solr will fetch this configuration from ZooKeeper when you request the collection creation. To upload the configuration, we use the `zkcli.sh` script provided with the Solr distribution. We use the `upconfig` command (the `-cmd` switch), which means that we want to upload the configuration. We specify the ZooKeeper host using the `-zkHost` property. After that, we can say which directory our configuration is stored (the `-confdir` switch). The directory should contain all the needed configuration files such as `schema.xml`, `solrconfig.xml`, and so on. Finally, we specify the name under which we want to store our configuration using the `-confname` switch.

After we have our configuration in ZooKeeper, we can create the collection. We do this by running a command to the Collections API that is available at the `/admin/collections` endpoint. First, we tell Solr that we want to create the collection (`action=CREATE`) and that we want our collection to be named `firstCollection` (`name=firstCollection`). Remember that the collection names are case sensitive, so `firstCollection` and `firstcollection` are two different collections. We specify that we want our collection to be built of two primary shards (`numShards=2`) and we want each shard to be present in two copies (`replicationFactor=2`). This means that we will have a primary shard and a single replica. Finally, we specify which configuration should be used to create the collection by specifying the `collection.configName` property.

As we can see in the cloud, a view of our cluster has been created and spread across all the nodes.

There's more...

There are a few things that I would like to mention—the possibility of running a Zookeeper server embedded into Apache Solr and specifying the Solr server name.

Starting an embedded ZooKeeper server

You can also start an embedded Zookeeper server shipped with Solr for your test environment. In order to do this, you should pass the `-DzkRun` parameter instead of `-DzkHost=192.168.0.10:9983`, but only in the command that sends our configuration to the Zookeeper cluster. So the final command for Solr 4.x should look similar to this:

```
java -DzkRun -jar start.jar
```

In Solr 5.0, the same command will be as follows:

```
bin/solr start -c
```

By default, ZooKeeper will start on the port higher by 1,000 to the one Solr is started at. So if you are running your Solr instance on `8983`, ZooKeeper will be available at `9983`.

The thing to remember is that the embedded ZooKeeper should only be used for development purposes and only one node should start it.

Specifying the Solr server name

Solr needs each instance of SolrCloud to have a name. By default, that name is set using the IP address or the hostname appended with the port the Solr instance is running on and the `_solr` postfix. For example, if our node is running on `192.168.56.1` and port `8983`, it will be called `192.168.56.1:8983_solr`. Of course, Solr allows you to change that behavior by specifying the hostname. To do this, start using the `-Dhost` property or add the host property to `solr.xml`.

For example, if we would like one of our nodes to have the name of `server1`, we can run the following command to start Solr:

```
java -DzkHost=192.168.1.10:9983 -Dhost=server1 -jar start.jar
```

In Solr 5.0, the same command would be:

```
bin/solr start -c -h server1
```

Setting up multiple collections on a single cluster

Having a single collection inside the cluster is nice, but there are multiple use cases when we want to have more than a single collection running on the same cluster. For example, we might want users and books in different collections or logs from each day to be only stored inside a single collection. This recipe will show you how to create multiple collections on the same cluster.

Getting ready

Before reading further, I advise you to read the *Creating a new SolrCloud cluster* recipe of this chapter. This recipe will show you how to create a new SolrCloud cluster. We also assume that ZooKeeper is running on `192.168.1.10` and is listening on port `2181` and that we already have four SolrCloud nodes running as a cluster.

How to do it...

As we already have all the prerequisites, such as ZooKeeper and Solr up and running, we need to upload our configuration files to ZooKeeper to be able to create collections:

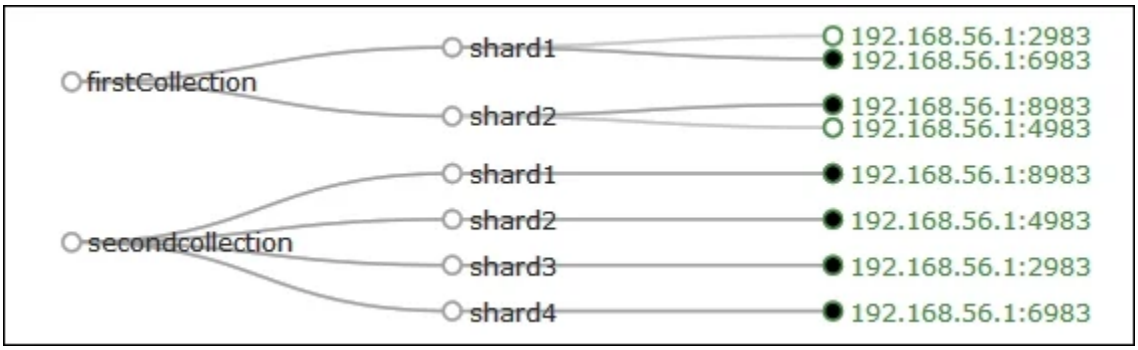
1. Assuming that we have our configurations in `/home/conf/firstcollection/conf` and `/home/conf/secondcollection/conf`, we will run the following commands from the `home` directory of the first run Solr server to upload the configuration to ZooKeeper (the `zkcli.sh` script can be found in the Solr deployment example in the `scripts/cloud-scripts` directory):

```
./zkcli.sh -cmd upconfig -zkhost localhost:2181 -confdir
/home/conf/firstcollection/conf/ -confname firstcollection
./zkcli.sh -cmd upconfig -zkhost localhost:2181 -confdir
/home/conf/secondcollection/conf/ -confname secondcollection
```

2. We have pushed our configurations into Zookeeper, so now we can create the collections we want. In order to do this, we use the following commands:

```
curl 'localhost:8983/solr/admin/collections?
action=CREATE&name=firstCollection&numShards=2&replicationFactor=2&collection.c
curl 'localhost:8983/solr/admin/collections?
action=CREATE&name=secondcollection&numShards=4&replicationFactor=1&collection.
```

3. Now, just to test whether everything went well, we will go to `http://localhost:8983/solr/#/~cloud`. As the result, we will see the following cluster topology:



As we can see, both the collections were created the way we wanted. Now let's see how that happened.

How it works...

We assume that we already have ZooKeeper installed—it is empty and doesn't have information about any collections, because we didn't create them. We also assumed that we have our SolrCloud cluster configured and started.

To create collections, we start by uploading the configuration to ZooKeeper. This is what we already discussed in the *Creating a new SolrCloud cluster* recipe of this chapter. We start by uploading two configurations to ZooKeeper, one called `firstcollection` and the other called `secondcollection`. After that, we are ready to create our collections.

We start by creating the collection named `firstCollection` that is built of two primary shards and one replica. The second collection, called `secondcollection`, is built of four primary shards and it doesn't have any replicas. We can see that easily in the cloud view of the deployment. The `firstCollection` collection has two shards— `shard1` and `shard2`. Each of the shards has two physical copies—one green (which means active) and one with a black dot, which is the primary shard. The `secondcollection` collection is built of four physical shards—each shard has a black dot near its name, which means that they are primary shards.

Splitting shards

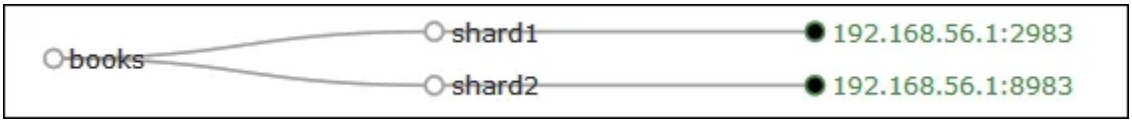
Imagine a situation where you reach a limit of your current deployment—the number of shards is just not enough. For example, the indexing throughput is lower and lower, because the disks are not able to keep up. Of course, one of the possible solutions is to spread the index across more shards; however, you already have a collection and you want to keep the data and reindexing is not an option, because you don't have the original data. Solr can help you with such situations by allowing splitting shards of already created collections. This recipe will show you how to do it.

Getting ready

Before reading further, I would suggest you all to read the *Creating a new SolrCloud cluster* recipe of this chapter. This recipe will show you how to create a new SolrCloud cluster. We also assume that ZooKeeper is running on `192.168.1.10` and is listening on port `2181` and that we already have four SolrCloud nodes running as a cluster.

How to do it...

Let's assume that we already have a SolrCloud cluster up and running and it has one collection called `books`. So our cloud view (which is available at `http://localhost:8983/solr/#/~cloud`) looks as follows:



We have four nodes and we don't utilize them fully. We can say that these two nodes in which we have our shards are almost fully utilized. What we can do is create a new collection and reindex the data or we can split shards of the already created collection. Let's go with the second option:

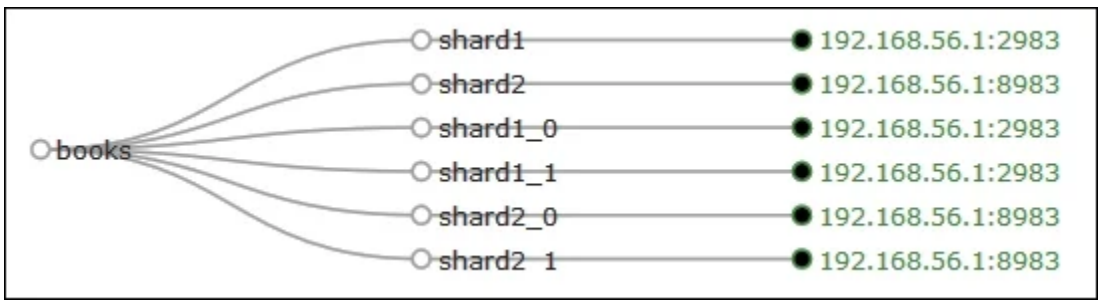
1. We start by splitting the first shard. It is as easy as running the following command:

```
curl 'http://localhost:8983/solr/admin/collections?
action=SPLITSHARD&collection=books&shard=shard1'
```

2. After this, we can split the second shard by running a similar command to the one we just used:

```
curl 'http://localhost:8983/solr/admin/collections?
action=SPLITSHARD&collection=books&shard=shard2'
```

3. Let's take a look at the cluster cloud view now (which is available at `http://localhost:8983/solr/#/~cloud`):

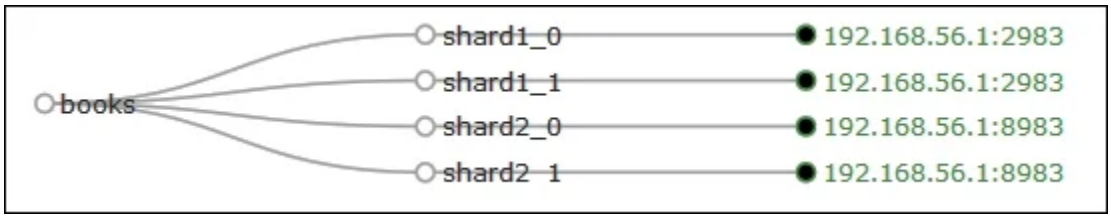


As we can see, both shards were split— `shard1` was divided into `shard1_0` and `shard1_1` and `shard2` was divided into `shard2_0` and `shard2_1`. Of course, the data was copied as well, so everything is ready.

However, the last step should be to delete the original shards. Solr doesn't delete them, because sometimes applications use shard names to connect to a given shard. However, in our case, we can delete them by running the following commands:

```
curl 'http://localhost:8983/solr/admin/collections?
action=DELETESHARD&collection=books&shard=shard1'
curl 'http://localhost:8983/solr/admin/collections?
action=DELETESHARD&collection=books&shard=shard2'
```

Now if we would again look at the cloud view of the cluster, we will see the following:



How it works...

We start with a simple collection called `books` that is built of two primary shards and no replicas. This is the collection which shards we will try to divide it without stopping Solr.

Splitting shards is very easy. We just need to run a simple command in the Collections API (the `/admin/collections` endpoint) and specify that we want to split a shard (`action=SPLITSHARD`). We also need to provide additional information such as which collection we are interested in (the `collection` parameter) and which shard we want to split (the `shard` parameter). You can see the name of the shard by looking at the cloud view or by reading the cluster state from ZooKeeper. After sending the command, Solr might force us to wait for a substantial amount of time—shard splitting takes time, especially on large collections. Of course, we can run the same command for the second shard as well.

Finally, we end up with six shards—four new and two old ones. The original shard will still contain data, but it will start to re-route requests to newly created shards. The data was split evenly between the new shards. The old shards were left although they are marked as inactive and they won't have any more data indexed to them. Because we don't need them, we can just delete them using the `action=DELETESHARD` command sent to the same Collections API. Similar to the `split shard` command, we need to specify the collection

name, which shard we want to delete, and the name of the shard. After we delete the initial shards, we now see that our cluster view shows only four shards, which is what we were aiming at.

We can now spread the shards across the cluster, and we do this in the *Moving shards between nodes* recipe later in this chapter.

Having more than a single shard from a collection on a node

Let's assume that we have a cluster of four nodes, but we are sure that we will have eight nodes in the near future. We want to be prepared for this, so when the new nodes are installed, we don't need to split collection shards or reindex the data. By default, Solr won't allow you to do that, but this recipe will show you how to achieve such a setup.

Getting ready

Before reading further, I would suggest you all to read the *Creating a new SolrCloud cluster* recipe in this chapter. This recipe will show you how to create a new SolrCloud cluster. We also assume that ZooKeeper is running on `192.168.1.10` and is listening on port `2181` and that we already have four SolrCloud nodes running as a cluster.

How to do it...

Let's assume that we already have a cluster with four nodes and the configuration uploaded to ZooKeeper (if you don't know how to do it, take a look at the *Creating a new SolrCloud cluster* recipe in this chapter). Our collection configuration is called `firstcollection` .

As we said, we have four nodes, but we would like to create a collection that has eight shards. We want to be prepared for cluster growth, which will happen in the future, and we don't want to use shard splitting, because we know it will take time and that operation is resource intensive. What we need to do is force Solr to put more than a single shard of the collection on a node and create a collection that is built of eight shards:

1. Let's try to create a new collection by running the following command:

```
curl 'localhost:8983/solr/admin/collections?
action=CREATE&name=firstTryCollection&numShards=8&replicationFactor=1&collectio
```

After running this command, we will see the following response from Solr:

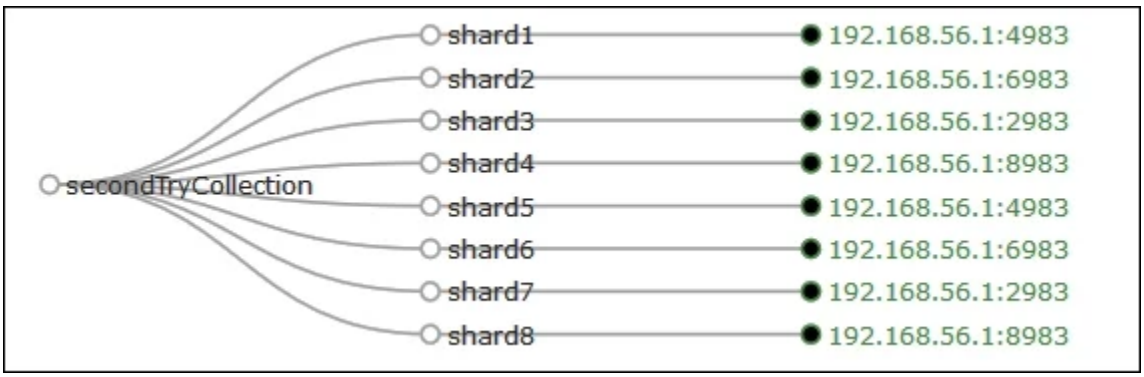
```
<?xml version="1.0" encoding="UTF-8"?>
<response>
<lst name="responseHeader"><int name="status">400</int><int
name="QTime">74</int></lst><str name="Operation createcollection caused
exception:">org.apache.solr.common.SolrException:org.apache.solr.common.SolrExc
Cannot create collection firstTryCollection. Value of maxShardsPerNode
is 1, and the number of live nodes is 4. This allows a maximum of 4 to be
created. Value of numShards is 8 and value of replicationFactor is 1. This
requires 8 shards to be created (higher than the allowed number)</str><lst
name="exception"><str name="msg">Cannot create collection firstTryCollection.
Value of maxShardsPerNode is 1, and the number of live nodes is 4. This allows
a maximum of 4 to be created. Value of numShards is 8 and value of
replicationFactor is 1. This requires 8 shards to be created (higher than the
allowed number)</str><int name="rspCode">400</int></lst><lst name="error"><str
name="msg">Cannot create collection firstTryCollection. Value of
maxShardsPerNode is 1, and the number of live nodes is
4. This allows a maximum of 4 to be created. Value of numShards is 8 and value
of replicationFactor is 1. This requires 8 shards to be created (higher than
the allowed number)</str><int name="code">400</int></lst>
</response>
```

This means that Solr won't create the collection, because there are not enough Solr nodes.

2. Let's now add `maxShardsPerNode=2` to our command so that it looks as follows:

```
curl 'localhost:8983/solr/admin/collections?
action=CREATE&name=secondTryCollection&numShards=8&replicationFactor=1&collecti
```

3. If we now take a look at `http://localhost:8983/solr/#/~cloud`, we will see the following:



As we can see, it worked. Let's now see why.

How it works...

Of course, we assume that we have our four nodes up and running and we have a collection configuration stored in ZooKeeper under the name `firstcollection`.

When we run the first command, we tell Solr that we want our `firstTryCollection` collection to be created with eight shards (`numShards=8`) and with only a single physical copy of each shard (`replicationFactor=1`), so no replicas. However, Solr returned with an error telling us that we can't do that. That is because, by default, Solr allows only a single shard of the same collection to be present on a given node. The default behavior is good, but it is not what we want in some use cases.

Because of this, we introduced the `maxShardsPerNode` property and we set it to `2`. The default value for that property is `1`, but in our case, we have four nodes and we create a collection built of eight shards, which means that we need at least two shards of the same collection on a single node. As we can see, after introducing this parameter and sending the command to create a collection, we ended up with a collection created.

Now, after our new nodes arrive, we will be able to spread the shards across the cluster. You will learn how to do this in the *Moving shards between nodes* recipe later in this chapter.

Creating a collection on defined nodes

There are use cases where we want to create a collection only on some of the nodes. For example, we would like to have one collection created on other better machines, because we know that this collection will be heavier in terms of indexing and querying. The other collections can live on the smaller and worse performing nodes. In such cases, we still have all the collections in a single cluster, but we know that they will be placed on hardware we want them to be placed. This recipe will show you how to do this.

Getting ready

Before reading further, I advise you to read the *Creating a new SolrCloud cluster* and *Having more than a single shard from a collection on a node* recipes of this chapter. These recipes will show you how to create a new SolrCloud cluster. We also assume that ZooKeeper is running on `192.168.1.10` and is listening on port `2181` and that we already have four SolrCloud nodes running as a cluster. We also assume that we have a configuration called `firstCollection` stored in ZooKeeper.

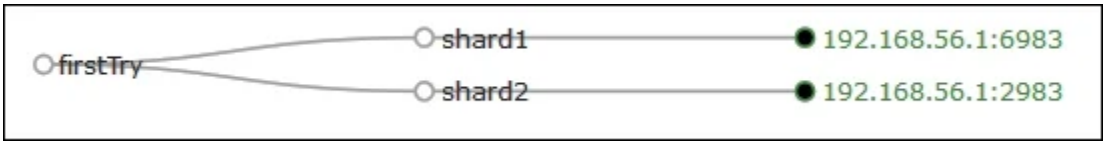
How to do it...

Let's assume that we have a cluster of four SolrCloud nodes and, as we wrote, we would like the primary shards to only be present on the better machines—in our case, these are the nodes called `192.168.56.1:8983_solr` and `192.168.56.1:7983_solr` :

1. If we run a standard collection creation command and specify that we will allow two shards of that collection per node, we will use the following command:

   ```
   curl 'localhost:8983/solr/admin/collections?
   action=CREATE&name=firstTry&numShards=2&replicationFactor=1&collection.configNa
   ```

   It might happen that our collection has been created the way we wanted, but let's check this by taking a look at `http://localhost:8983/solr/#/~cloud` . This is what we see:

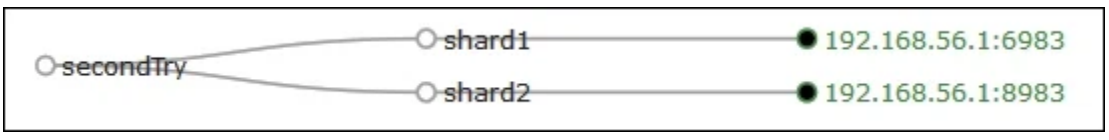
   

   As you can see, it is not exactly what we wanted.

2. So let's try specifying the nodes we want our collection to be created at by adding the `createNodeSet` property. Our new command looks as follows:

   ```
   curl 'localhost:8983/solr/admin/collections?
   action=CREATE&name=secondTry&numShards=2&replicationFactor=1&collection.configN
   ```

3. If we now again take a look at `http://localhost:8983/solr/#/~cloud` , we will see the following cluster view:

As we can see, everything worked as we wanted. Let's now see why that happened.

How it works...

We assume that the better nodes are the ones that have the names of `192.168.56.1:8983_solr` and `192.168.56.1:6983_solr`. When we run our standard collection creation command, we can end up with collections created on those nodes; however, we can't count on that. In fact, in our example, we ended up with collections created on nodes `192.168.56.1:2983_solr` and `192.168.56.1:6983_solr`. This is not what we wanted.

To achieve what we want, we need to add the `createNodeSet` property. The value of this property should be a comma-separated list of SolrCloud node names, which should be taken into consideration when creating collections. In our case, the value of the `createNodeSet` property should be `192.168.56.1:8983_solr,192.168.56.1:6983_solr`. As we can see, after we added this property, our collection named `secondTry` was properly created only on the nodes we were interested in.

Adding replicas after collection creation

Replicas are copies of the primary shard. They are very useful when it comes to handing performance issues—for example, if your current cluster can't keep up with queries, you can add new nodes and increase the number of replicas. This way, more Solr servers can serve queries and each of them will have fewer queries to process. In this recipe, we will learn how to add new shards to already created collections.

Getting ready

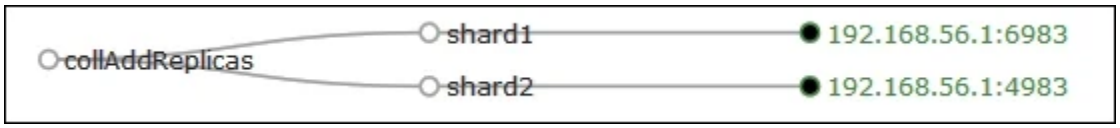
Before reading further, I advise you to read the *Creating a new SolrCloud cluster* and *Having more than a single shard from a collection on a node* recipes of this chapter. These recipes will show you how to create a new SolrCloud cluster and a collection. We also assume that ZooKeeper is running on `192.168.1.10` and is listening on port `2181`. We already have the configuration called `firstcollection` stored in ZooKeeper and we already have four SolrCloud nodes running as a cluster.

How to do it...

1. Let's start by creating a sample collection with two shards and no replicas. We do this by running the following command:

```
curl 'localhost:8983/solr/admin/collections?
action=CREATE&name=collAddReplicas&numShards=2&replicationFactor=1&collection.c
```

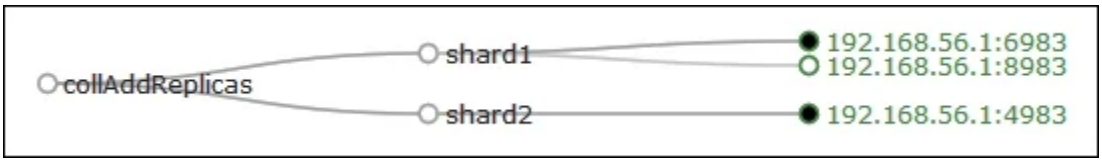
After running the preceding command, our cluster view should look as follows:



2. Now, let's add a replica of `shard1` and let's place it on a node called `192.168.56.1:8983_solr`. To do this, we will run the following command:

```
curl 'http://localhost:8983/solr/admin/collections?
action=ADDREPLICA&shard=shard1&collection=collAddReplicas&node=192.168.56.1:898
```

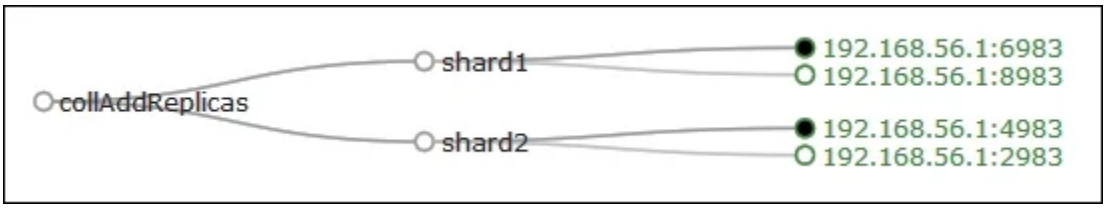
After Solr is done working, our cluster view should look similar to the following screenshot:



3. Now, let's add a replica of `shard2` and let's place it on a node called `192.168.56.1:2983_solr`. To do this, we will run the following command:

```
curl 'http://localhost:8983/solr/admin/collections?
action=ADDREPLICA&shard=shard2&collection=collAddReplicas&node=192.168.56.1:298
```

4. If we now again look at the cluster view, we will be able to see something similar to this:



As we can see, everything works as it should.

How it works...

We started by creating a new collection called `collAddReplicas`. Of course, we have our SolrCloud cluster up and running, ZooKeeper installed and running, and the `firstcollection` configuration stored in it. Our collection is created with two shards (`numShards=2`) and with only primary shards (`replicationFactor=1`).

As we said, after some time, we added new nodes to our cluster because our initial cluster was heavily used and the queries started to be slow. Because of that, we decided to add more replicas.

We started by creating a replica for `shard1`. To do this, we need to call the Collections API `ADDREPLICA` command (`action=ADDREPLICA`) and specify the shard we want to create the replica for (`shard=shard1` in our case), the collection that this command should modify (`collection=collAddReplicas`), and the node name on which the replica should be created (`node=192.168.56.1:8983_solr`). After receiving this command, Solr will create that replica and start replication. After replication has ended, Solr will automatically start using that replica. We do the same for `shard2`—although we put the replica on a different node.

As we can see, after adding two replicas, we now have four shards in our collection, two for each shard, and Solr successfully initiated our new shards.

Removing replicas

Sometimes, there is a need of removing one or more replicas from your Solr cluster. Either because you want to get rid of some nodes and you want your cluster state to become clean or you want to move some shard to another node and delete the original shard. No matter what the case is, there might come a time where you need to delete a shard. This recipe will show you how to do it.

Getting ready

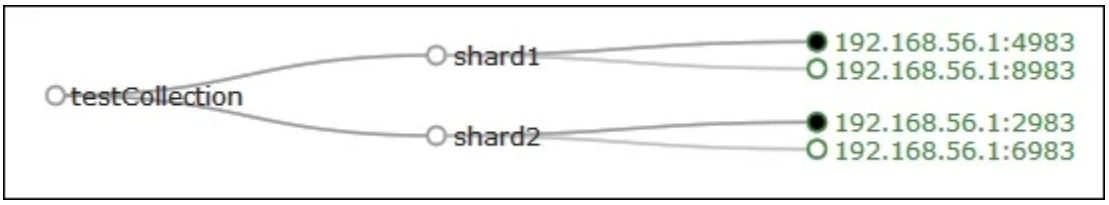
Before reading further, I advise you to read the *Creating a new SolrCloud cluster* and *Having more than a single shard from a collection on a node* recipes in this chapter. These recipes will show you how to create a new SolrCloud cluster and create a collection. We also assume that ZooKeeper is running on `192.168.1.10` and is listening on port `2181`. We already have the configuration called `firstcollection` stored in ZooKeeper and we already have four SolrCloud nodes running as a cluster.

How to do it...

1. To show you how to delete replicas from the already existing collection, we will create a new collection that is built of two shards and have a replica of each of the shards. To do this, we will run the following command:

   ```
   curl 'localhost:8983/solr/admin/collections?
   action=CREATE&name=testCollection&numShards=2&replicationFactor=2&collection.co
   ```

   After the collection is created, we should see the following cluster view when we take a look at `http://localhost:8983/solr/#/~cloud`:

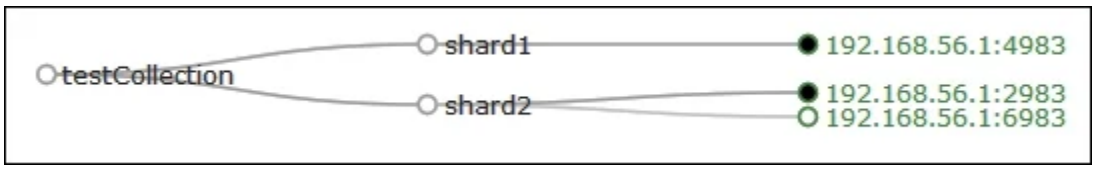
   

   Now, let's assume that we want to get rid of two servers in our cluster, `192.168.56.1:8983` and `192.168.56.1:6983`. And we don't want to leave any traces in the cluster state. To do this, we need to remove the replicas that are placed on those nodes.

2. We start by removing the `192.168.56.1:8983_solr` replica by running the following command:

   ```
   curl 'http://localhost:8983/solr/admin/collections?
   action=DELETEREPLICA&collection=testCollection&shard=shard1&replica=core_node2'
   ```

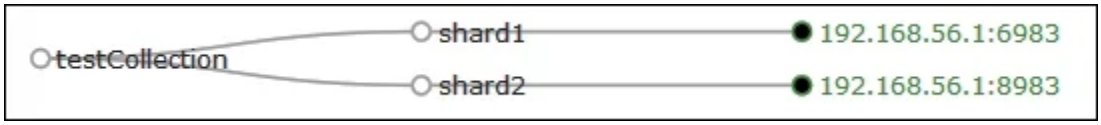
3. After the command is executed, we can see the following cluster view:

4. We can now remove the replica of the second shard by running the following command:

```
curl 'http://localhost:8983/solr/admin/collections?
action=DELETEREPLICA&collection=testCollection&shard=shard2&replica=core_node4'
```

And again let's take a look at the cluster view:



As we can see, everything worked.

How it works...

We start by creating a collection that we will use to demonstrate how to remove replicas from the already created collection. Of course, we have our SolrCloud cluster up and running, ZooKeeper installed and running, and the `firstcollection` configuration stored in it. The `collection creation` command was already mentioned a few times—in our case, we created a collection called `testCollection` that is built of four physical shards—two primary shards and one replica for each of them.

What we try to do is delete the replicas of the primary shards, because we want to remove the server on which they are running and throw the hardware away. To do this, we start by removing the replica that is stored on the node running on port `8983` . The thing is that this time we don't specify the node name like during replica creation, but we need to specify the actual shard name that Solr uses internally. We can do this by taking a look at `clusterstate.json` in the Solr admin panel (you can do that by going to `http://localhost:8983/solr/#/~cloud?view=tree` with your web browser and choosing the mentioned file). In our case, that file looks as follows:

```
{"testCollection":{
    "shards":{
      "shard1":{
        "range":"80000000-ffffffff",
        "state":"active",
        "replicas":{
          "core_node1":{
            "state":"active",
            "core":"testCollection_shard1_replica1",
            "node_name":"192.168.56.1:4983_solr",
            "base_url":"http://192.168.56.1:4983/solr",
            "leader":"true"},
          "core_node2":{
            "state":"active",
            "core":"testCollection_shard1_replica2",
            "node_name":"192.168.56.1:8983_solr",
            "base_url":"http://192.168.56.1:8983/solr"}}},
      "shard2":{
        "range":"0-7fffffff",
        "state":"active",
        "replicas":{
          "core_node3":{
            "state":"active",
            "core":"testCollection_shard2_replica1",
            "node_name":"192.168.56.1:2983_solr",
            "base_url":"http://192.168.56.1:2983/solr",
            "leader":"true"},
          "core_node4":{
            "state":"active",
            "core":"testCollection_shard2_replica2",
            "node_name":"192.168.56.1:6983_solr",
            "base_url":"http://192.168.56.1:6983/solr"}}}},
    "maxShardsPerNode":"1",
    "router":{"name":"compositeId"},
    "replicationFactor":"2",
    "autoAddReplicas":"false"}}
```

For example, for `shard1`, we have two replicas, one named `core_node1` and the second one named `core_node2`. The first one is the primary shard (`leader=true`). For `shard2`, we also have two replicas, one named `core_node3` and the second one named `core_node4`. In this case, the first is also the primary shard.

To delete a shard, we need to send an appropriate request to the Collections API (`/admin/collections`) and specify the delete action (`action=DELETEREPLICA`), collection (`collection=testCollection`), the shard (`shard=shard1`), and the name of the replica (`replica=core_node2`). In our case, we want to remove the replica from nodes running on port `8983` for `shard1` (which means that we remove `core_node4`) and on port `6983` for `shard2` (which means we remove `core_node4`).

As we can see, after both the commands were executed, we are now left with a collection that has only primary shards. If we look at the cluster state now, we will see that there are no trace of the replicas, which means that Solr removed them properly.

Moving shards between nodes

There are moments where we want to move shards between nodes in SolrCloud cluster. Until now, the Solr Collections API doesn't have a command telling Solr to move a single shard to another node. We need to do such an operation manually. For example, let's assume that we want to exchange one of the nodes in our cluster with a new server, but we don't want any downtime or interruptions to our service. This recipe will show you how to do that.

Getting ready

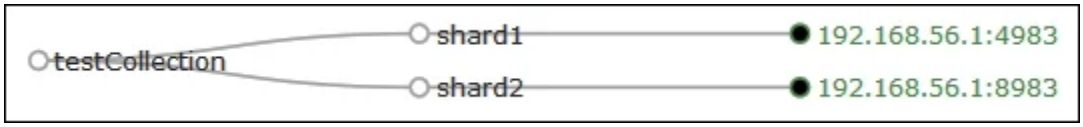
Before reading further, I would suggest you all to read the *Creating a new SolrCloud cluster*, *Adding replicas after collection creation*, and *Removing replicas* recipes of this chapter. These recipes will show you how to create a new SolrCloud cluster and create a collection. We also assume that ZooKeeper is running on `192.168.1.10` and is listening on port `2181`. We already have the configuration called `firstcollection` stored in ZooKeeper and we already have four SolrCloud nodes running as a cluster.

How to do it...

1. To keep things simple, we will start by creating a collection that is built of two shards and no replicas. We do this by running the following command:

   ```
   curl 'localhost:8983/solr/admin/collections?
   action=CREATE&name=testCollection&numShards=2&replicationFactor=1&collection.co
   ```

   Our cluster view will look as follows:

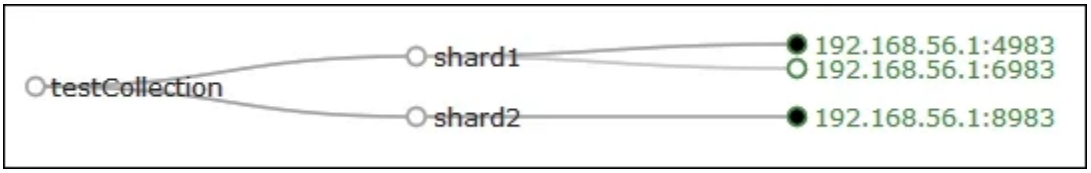


2. Now, let's try to move `shard1` from `192.168.56.1:4983` to `192.168.56.1:6983` (of course, we assume that we have that node in the cluster). We do this because we want to replace the server identified by Solr as `192.168.56.1:4983` with the one identified as `192.168.56.1:6983` .

3. To do this, we need to create a new replica on the `192.168.56.1:6983` node; we do that by running the following command:

```
curl 'http://localhost:8983/solr/admin/collections?
action=ADDREPLICA&shard=shard1&collection=testCollection&node=192.168.56.1:6983
```

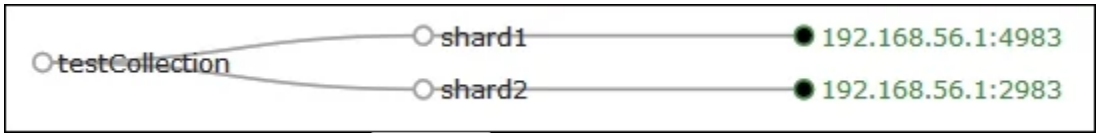
   After the command is completed and Solr synchronizes the replica, the cluster view should look as follows:



4. Now we need to remove the primary shard of `shard1` —the one on `192.168.561:8983` . We do this by running the following command:

```
curl 'http://localhost:8983/solr/admin/collections?
action=DELETEREPLICA&collection=testCollection&shard=shard1&replica=core_node2'
```

5. After the command succeeds, we can see the following cluster view:



As we can see, the primary shard of `shard1` is now located on the node `192.168.56.1:6983` , which is what we wanted.

How it works...

We start by creating a collection that we will use to demonstrate the migration of a shard from one node to another. Of course, we have our SolrCloud cluster and ZooKeeper installed and running and also have the `irstcollection` configuration stored in it. The `collection creation` command was already mentioned a few times—in our case, we created a collection called `testCollection` that is built of four physical shards—two primary shards and one replica for each of them.

As we said, we want to move `shard1` from the node `192.168.56.1:4983` to node `192.168.56.1:6983` . To do this, we need to create a new shard first—a replica of the shard we want to migrate on the node we want our shard to be migrated. In our case, we

created a replica of `shard1` on the `192.168.56.1:6983` node by running an `ADDREPLICA` command using the Collections API (if you are not familiar with the `add replica` command, refer to the *Adding replicas after collection creation* recipe in this chapter).

After Solr created the replica and finished replicating the data, we are now ready to delete the original shard. In some cases, it would be good to stop indexing and wait for the data in the primary shard to be in the same state as the data on the replica. This will ensure us that no data loss will happen. After we know Solr is ready, we can just run the `DELETEREPLICA` command and remove the primary shard (if you are not familiar with the `delete replica` command, refer to the *Removing replicas* recipe in this chapter). After the delete command succeeds, we end up with collection built of two primary shards, one left intact and the second one moved to a new node. We can now turn off the old node and still have the cluster state clean. Of course, we should wait for the replication to finish (we can check the status in the Solr administration UI).

Using aliasing

Aliasing is the functionality of giving your collections more than a single name. It might seem not very useful, but in fact this is very useful—for example, when dealing with time series data. Imagine that you have data that stores logfiles and as the data is so large, you create a new collection every day. In addition to this, you usually search for the latest or one week worth of data. To simplify indexing and searching, we can use aliasing, and this recipe will show you how to do that.

Getting ready

Before reading further, I would advise you all to read the *Creating a new SolrCloud cluster* recipe of this chapter. This recipe will show you how to create a new SolrCloud cluster and create a collection. We also assume that ZooKeeper is running on `192.168.1.10` and is listening on port `2181`. We already have the configuration called `firstcollection` stored in ZooKeeper and we already have four SolrCloud nodes running as a cluster.

How to do it...

1. When we created our cluster, we started with a single collection that was created using the following command:

```
curl 'localhost:8983/solr/admin/collections?action=CREATE&name=logs_2014-09-01&numShards=2&replicationFactor=1&collection.configName=firstcollection'
```

2. We then created our daily collection and now we need to add two aliases—one for indexing and searching today's data (called `today`) and one for searching last week's data (called `lastWeek`). We do this by running the following commands:

```
curl 'localhost:8983/solr/admin/collections?action=CREATEALIAS&name=today&collections=logs_2014-09-01'
curl 'localhost:8983/solr/admin/collections?action=CREATEALIAS&name=lastWeek&collections=logs_2014-09-01'
```

   If we take a look at the tree view in the Solr admin panel and the `aliases.json` file, we will see the following (you can do this by going to `http://localhost:8983/solr/#/~cloud?view=tree` with your web browser and choosing the mentioned file):

```
{"collection":{
    "lastWeek":"logs_2014-09-01",
    "today":"logs_2014-09-01"}}
```

3. After the day has ended, we create a new daily collection by running the following command:

```
curl 'localhost:8983/solr/admin/collections?action=CREATE&name=logs_2014-09-02&numShards=2&replicationFactor=1&collection.configName=firstcollection'
```

4. Once done, we need to modify our aliases. We also need the `today` alias to point the newly created collection and the `lastWeek` alias to cover two collections. We do this by running the following commands:

```
curl 'localhost:8983/solr/admin/collections?action=CREATEALIAS&name=today&collections=logs_2014-09-02'
curl 'localhost:8983/solr/admin/collections?action=CREATEALIAS&name=lastWeek&collections=logs_2014-09-02,logs_2014-09-01'
```

5. If we again look at the `aliases.json` file, we will see the following:

```
{"collection":{
    "lastWeek":"logs_2014-09-02,logs_2014-09-01",
    "today":"logs_2014-09-02"}}
```

Of course, everything worked as it should and we can easily run queries and indexing requests. For example, to take a look at the data from last week, we will run the following command:

```
curl 'localhost:8983/solr/lastWeek/select?q=*:*'
```

Now we can repeat the same steps daily and can continue using aliases, which will help you to change a script or application every day.

How it works...

We started off by creating our first daily collection called `logs_2014-09-01`. This will be used to index logs from September 1. Because we don't want our application to be forced to roll the indices, we create aliases. The alias called `today` will be used for indexing and searching the newest data; the second alias called `lastWeek` will be only used for searching. One thing to remember is that we can use aliases for indexing only if an alias points to a single index—which is true for our `today` alias.

However, let's get back to the commands used to create the aliases. Of course, we call the Collections API (`/admin/collections`) and we tell Solr that we want an alias to be created (`action=CREATEALIAS`). We specify the name of the alias using the `name` property and finally we specify the name of collections (using the `collections` parameter) separated by the comma character that will be covered by that alias. After the command execution ends, you are allowed to use both the alias and the collection name.

When the day ended, we changed the aliases. The one called `today` was changed to the new collection—the `logs_2014-09-02` one—and the second one called `lastWeek` was also changed to include the newest collection. Note that Solr will overwrite the previously created alias if we request to create an alias with a name already present.

Using routing

Routing is the ability to point queries and index it to a single shard of the collection. Let's assume that we have a single collection and store data of hundreds of clients in that collection. We can have a single collection per customer, but there are just too many of them, so such a solution is not scalable at all. Instead we go with one collection and we keep

the data of a single customer in a single shard, so when querying we don't have to query all the shards. This allows you to save resources when querying. This recipe will show you how to do it.

Getting ready

Before reading further, I advise you to read the *Creating a new SolrCloud cluster* recipe of this chapter. This recipe will show you how to create a new SolrCloud cluster and create a collection. We also assume that ZooKeeper is running on `192.168.1.10` and is listening on port `2181` and we already have four SolrCloud nodes running as a cluster.

How to do it...

1. For the purpose of this recipe, we will use the following index structure (we need to add the following section to our `schema.xml` file):

```
<field name="id" type="string" indexed="true" stored="true" required="true" />
<field name="title" type="text_general" indexed="true" stored="true" />
<field name="customer" type="string" indexed="true" stored="true" />
```

2. For a customer named as `customer_1` , we have the following data (stored in a file called `data_customer_1.xml`):

```
<add>
 <doc>
  <field name="id">customer_1!1</field>
  <field name="title">Customer document 1</field>
  <field name="customer">customer_1</field>
 </doc>
 <doc>
  <field name="id">customer_1!2</field>
  <field name="title">Customer document 2</field>
  <field name="customer">customer_1</field>
 </doc>
</add>
```

3. For a customer named as `customer_2` , we have the following data (stored in a file called `data_customer_2.xml`):

```
<add>
 <doc>
  <field name="id">customer_2!3</field>
  <field name="title">Customer document</field>
  <field name="customer">customer_2</field>
 </doc>
</add>
```

4. We start by uploading our collection configuration to ZooKeeper. Assuming that we have our configuration in `/home/configuration/customers/conf`, we run the following command from the `home` directory of the first run Solr server (the `zkcli.sh` script can be found in the example Solr deployment in the `scripts/cloud-scripts` directory):

```
./zkcli.sh -cmd upconfig -zkhost 192.168.1.10:9983 -confdir
/home/configuration/customers/conf/ -confname customersConfig
```

5. After this, we can create our collection by running the following command:

```
curl 'localhost:8983/solr/admin/collections?
action=CREATE&name=customers&collection.configName=customersConfig&numShards=4&
```

6. Now we can start indexing our data—we do this in the same way that we would index the data without routing. For example, to index the two files, we will use the following commands:

```
curl 'http://localhost:8983/solr/customers/update' --data-binary
@data_customer_1.xml -H 'Content-type:application/xml'
curl 'http://localhost:8983/solr/customers/update' --data-binary
@data_customer_2.xml -H 'Content-type:application/xml'
curl 'http://localhost:8983/solr/customers/update' --data-binary '<commit/>' -H
'Content-type:application/xml'
```

7. Now let's try to query the data. For example, if we would like to only get data for a customer named as `customer_2`, we will run a standard query with the `_route_` property set to the routing value for that customer. We also need a proper filter query. So for a customer named as `customer_2`, the following command will be used:

```
curl 'localhost:8983/solr/customers/select?
q=*:*&fq=customer:customer_2&_route_=customer_2!'
```

The response returned by Solr will be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
 <lst name="responseHeader">
  <int name="status">0</int>
  <int name="QTime">13</int>
  <lst name="params">
   <str name="q">*:*</str>
   <str name="fq">customer:customer_2</str>
   <str name="_route_">customer_2!</str>
  </lst>
 </lst>
 <result name="response" numFound="1" start="0" maxScore="1.0">
  <doc>
   <str name="id">customer_2!3</str>
   <str name="title">Customer document</str>
   <str name="customer">customer_2</str>
   <long name="_version_">1478847760611409920</long>
  </doc>
 </result>
</response>
```

As we can see, everything worked as it should.

How it works...

Our index structure is very simple. Each of the documents have three fields—one for the document identifier (the `id` field), one for the title of the document (the `title` field), and the customer field that will be used for filtering. What we have to look at is the data itself. Take a closer look at the identifiers of documents. Each identifier is prefixed with the customer name followed by the `!` character. When using composite routing (when no router is specified during collection creation), we can prefix the document identifier with a value and the `!` character after that, for example, `cookbook1234!1`. This means that Solr will use the `cookbook1234` value to determine in which shard the document will be indexed (by default, Solr uses the document identifier to determine that). The thing is that Solr will put the documents with the same routing value in the same shard. This means that data for a single customer will be put in the same shard in our case.

There is one additional thing to remember though—you usually end up with more routing values than what you have shards. In our case, we have four shards, but we might have hundreds of clients. Because of this, we need to remember about filtering during querying.

Next, we uploaded the configuration to ZooKeeper and finished creating the collection and then indexed our data. As you can see, there is nothing new in the indexation command. That's because nothing additional is needed while indexing apart from prefixing document identifiers with the routing value.

Finally, while querying, we need two things—the filter that will limit the data to only that customer (because we might have more than a single customer data in a single shard) and the second thing is the routing value. The routing value is the same as we used for prefixing document identifiers with the `!` character. We do this by adding the `_route_` property to our query. So for example, if we want to point our query to a shard where data for `customer_2` was indexed, we should add `_route_=customer_2!` to our query.

Also note that while using a composite router, Solr supports up to two levels of nesting, for example, `_rout e_=customer_2!USA!1` is allowed.